# Portfolio (POR) Assessment Part 1

**Assessment Resources:**

|  |
| --- |
| Marking key available for lecturer via Blackboard.  Students may refer to the lecture material in formulating their answers.  Material for this assessment is available:  *https://github.com/NM-TAFE/civ-ipos-sessions/tree/2023S2/assessments/POR/por-1* |

**Assessment Instructions:**

|  |
| --- |
| Students must attempt all questions. Answer succinctly using full sentences. At most two paragraphs are expected per answer.  All answers must be at the student’s own words – copying generated code or answers from ChatGPT or other AI tools is **strictly** prohibited.  **Please** **note**: you may have been invited to trial completing this assessment via GitHub classrooms. If you have, you do not need to also complete this assessment. |

# Overview

Tic-Tac-Toe Game Refactoring

## Objective

Refactor a given monolithic tic-tac-toe game, such that the code:

1. Is modular, consisting of at least two files that logically group related functions.
2. Implements an appropriate Python project folder structure.
3. Includes at least one test case.
4. Employs a 2D data structure.

In the process, you must use at least four functions, two classes, two files, and one import statement of your modules (not including imports used in the test case).

The source code you need to refactor can be found here:  
  
<https://github.com/NM-TAFE/civ-ipos-sessions/tree/2023S2/assessments/POR/por-1>

## Instructions

Complete each of the following steps carefully reviewing what is expected.

### Step 1: Review the Existing Code

Firstly, analyze the given tic-tac-toe game code. Understand the flow and functionality before proceeding with the refactoring.

### Step 2: Identify Components to Refactor

Identify the parts of the code that can be improved. Determine which parts of the code can be grouped *logically* into separate modules.

### Step 3: Modularizing the Code

Refactor the code to create at least **two** files. These files should contain logically grouped functions and/or classes. Ensure the file names are appropriate for the division you have chosen.

### Step 4: Create a Modern Python Folder Structure

The refactored code should adhere to the following Python folder structure:

tic\_tac\_toe/

|--- src/

| |--- \_\_init\_\_.py

| |--- module1.py

| |--- module2.py

|--- tests/

| |--- \_\_init\_\_.py

| |--- test\_tic\_tac\_toe.py

|--- setup.py

**Note**:

* \_\_init\_\_.py files are used to indicate that a directory should be treated as a Python package. This allows the files within to be imported as a module in the test scripts or other python files.
* setup.py is a Python file used to specify what modules and dependencies must be installed. The file has been provided for you, along with instructions on how to install your modules using this file. If PyCharm prompts you to run the setup.py – cancel – it will not work until you restructure your project!
* You must give your Python files appropriate names. Do **not** use module1, 2, etc.

### Step 5: Create a Test Case

Develop at least one test case for your refactored code. The test case should reside in the 'tests' directory. It is preferred that you use unittest framework for writing your test case.

### Step 6: Implement 2D Data Structure

Refactor the code such that it employs a 2D data structure for the tic-tac-toe game board.

### Step 7: Written Report

Once you have completed your refactoring, write a brief report addressing the following:

1. Justification for your refactoring decisions.
2. The challenges you would have faced maintaining and testing the original monolithic code.
3. How you would modify your refactored code to handle a custom-sized tic-tac-toe game (larger than 3x3), and how this implementation would be easier to handle than in the original code.

Justification:

I refactored the setting out of the board and its formatting to be code that is leaner and more automated. Instead of having a rigid game layout now it is scalable when the rows and columns are changed so will the board layout be done automatically.

Challenges:

A challenge with maintain the original program will be that the winning conditions and board set out are very rigid and if I was to change the board set up manually for each row and column and I would have to manually change the winning conditions adding each index location in the win\_conditons list.

A challenge with the testing the old code would be that the winning conditions are in a rigid list and that if the game board is set up differently it wouldn’t test the program accurately.

Approach to adding support for larger boards:

To add support for larger boards I will make the code for make formatting the board to be scalable so that when the rows and columns variable are changed the functions can handle it and create a game board automatedly that the user can define. To ensure all of the function and the game can work I will make it so a game board can only have one value to define it so that it as a square layout

### Step 8: Short Answer (Knowledge Questions)

Provide brief answers to the knowledge-question worksheet.

Briefly explain what is modular programming

Modular programming is the practice of breaking down a program in to self-contained modules that contain their own specific functionality such as in my program I have 2 python files (board\_functions and scoring\_system) in the .src folder they both contain functions specific to their name and are called to be used in the main.py program when playing the tic tac toe game

How can you import only a specific function or class from a module in Python? What is the syntax for this?

For example I have a program alter\_list.py inside this program is format\_list()

In another program is the same directory I can call it by

From alter\_list import format\_list

How would you explain Python's parameter-passing mechanism? Is it more like pass-by-value or pass-by-reference? Justify your answer.

Pythons parameter passing mechanism is more similar to pass by reference as python is an object based language it is referencing an object in memory when a value is called not the value itself.

Given the following Python code, what will be the output and why?

def modify\_list(list\_):

list\_.append("new")

list\_ = ["completely", "new"]

items = ["original"]

modify\_list(items)

print(items)

# Output:

The output will be [original , new]

# Explanation:

When the argument is call in the function it a list containing [original] then inside the function the list is appended with the value ‘new’. The list\_ variable is a local variable to the function and the line ‘list\_ = ["completely", "new"] creates a new list that is used anywhere.

In Python even though variables created within a function are local, there are still situations where you can modify data outside the scope with a local variable. Explain this anomaly and relate it to both mutability and pass by reference.

When an object is mutable it can be modified outside of its scope for example if the object is a list you can use your\_list.append to modify the object outside of its scope. You can pass a reference through a function for a mutable object and the object will modify the data outside of its scope where a non-mutable object it wouldn’t be possible.

List two benefits of modular coding approaches. How do these benefits assist in the development of medium-sized applications?

One of the benefits of modular coding is encapsulation by containing the inner workings of the functions / classes etc.. separate to other parts of the program. The functions can be easily be used but they are kept separate. Another benefit is that the modules are reusable and can be called if different parts of the program or even exported to another project

### Submission

Please submit the refactored code including the root folder but **excluding** the venv/ via Blackboard, also attach this worksheet. Note: if you are trialling the new GitHub classroom approach you only need to submit a note specifying that you completed the assessment.

Name your zip and document:

XXX\_source-por-part1.zip and XXX\_C-IPOS-AT1-POR-Part1.docx respectively.

Where XXX represents **your** two or three-character initials.

### Evaluation Criteria

Your refactoring will be evaluated on the clarity and modularity of your code, as well as the thoughtful reasoning behind your design decisions. Your test case should be robust and cover key aspects of the tic-tac-toe game functionality. The written report should accurately reflect your understanding of code refactoring, testing, and the flexibility of your new implementation.